أمثلة تطبيقية

مكتسبـــــات دورات JAVA

اسماعيل غلال

هذا السلسلة تقدم مجموعة من الأمثـلة على شكل تمارين محـــــــلولة تلخص مكتسبات دورات JAVA للأستاذ عبـد الله عيد في شبكته على الإنتــرنــــــت:

www.abdullaheid.net

أمثلة تطبيقية لمكتسبات دورات JAVA

© 2014 إسماعيل غلال جميع الحقوق محفوظة.

مقدمة

السلام عليكم و رحمة الله و بركاته ...

في ظل المـــــبادرة الكريمة للأستاذ عبد الله عيد بتسجيله لمجموعة كبيرة من الدروس لبعض أشهر لغــات البرمجة في موقعه www.abdullaheid.net، و بهذه المناسبة أود أن أشكره جزيل الشكر على صنيعه و وفائه، قررت أن أضع أمثلة إضافية علما أن هناك أمثـــــلة مرفقة في آخر كل الدورة (أقصد بها دورات JAVA)، لأنني أردت ألا أكون مستهلكـــا فقط ... المهم، في خلال قراءتك هذه السلسة (المتواضعة) ستتعرف على مجموعة من الـــدوال الجديدة التي تتعامل مع مختلف أنواع البيانات، و ستصادف دوال تعرفت عليها في الدورات، لكن الأهم هو تطبيقك لمـــــا اكتسبت منها، بالمناسبة هذه السلسة طبعا موجهة لكل من تابع دورات JAVA (بالأخص 101 و 102) و كل من له خلفية محترمة في لغة JAVA SE.

بالنسبة للتمارين ستكون على شكل إشكاليات برمجية، سيطلب منك حلها عبر كود برمجي، عند قراءتك للإشكالية لديك خيارين:

1- تتبع مراحل حلها، خطوة بخطوة. ☹

2- محاولة حلها بنفسك دون قراءة الحل المقترح. ☺

(طبعا هناك عدة طرق لكتابة كود برمجي لكن المهم هو أن نتيجة تنفيذه مطابقة للمطلوب منك ☺)

أما بالنسبة للسلسلة فستكون مقسمة لعدة كتب، كل كتاب يطرح تمرينا مختلفا، و يستهدف تعزيز دروس مختلفة.

إهداء

أهدي هذا الكتاب لكل عزيز على فـــؤادي:

عـــــائلتي الصغيرة و الكبيرة

الأستاذ عبد الله عيد

فريق مدى لبناء القدرات

كل إنسان يسعى لطلب العلم

كل إنسان يحب الخير و يسعى لفعل الخير

كل إنسان ☺

الفهرس

التعامل مع النصوص

الإشكالية رقم

1

كنت تبرمج برنامجا لتحرير النصوص يتميز بمجموعة من الخاصيات الرائعة، منها خاصية البحث عن كلمة أو عبارة في النص المكتوب.

اكتب برمجيا مبدأ عمل هذه الخاصية.

I. تحليل الإشكالية المطروحة

✓ تذكير :

الكونستراكتور (constructor) هو دالة تستدعى أثناء وقت إنشاء الكائن و غالبـــا ما تستعمل لتمهيد قيم المتغيرات المعرفة.	
البارامیتر (parameter) هو قيمة تمرر لدالة أثناء استدعائها .	

✓ إضاءة:

لن نكتب البرنامج كاملا (برنامج تحرير النصوص) بل فقط خاصية البحث.

✓ محاكاة خاصية البحث:

SearchProperty : اسم الكلاس (class)

SearchProperty() : كونستراكتور (constructor) بدون بـــارميترات (parameters).

دوره هو تحديد قيمة افتراضية للنص المكتوب (text).

SearchProperty(String ptext) : كونستراكتور (constructor) يأخذ بـــــــارميتر (parameter) واحد نوعه نص (String) سيتم وضع قيمته لـلــــــــــنص المكتوب (text).

search(String pw) : دالـــة (method) تأخذ بارميتر (parameter) واحد نوعه نص.

دورها هو البحث عن الكلمة (pw) في النص المكتوب (text) و هي لا ترجع أي شيء.

String text : متغير (variable) نوعه نص (String) يمثل النص المكتوب.

II. الكود البرمجي

✓ كلاس SearchProperty:

```
package examples ;
public  class  SearchProperty {
        String  text ;
```

```
        public  SearchProperty ( ) {


        }
        public  SearchProperty ( String ptext ) {


        }
        public  void  search ( String pw ) {


        }
}
```

✓ **ملحوظة مهمة:**

هناك طبعا عدة طرق لكتابة هذا الكلاس (**class**) لكني فضلت هذه الطريقة للتطرق لأغلب حالات استعمال النصوص (تمرير النص كبارميتر (**parameter**) – تعيين نص كقيمة لمتغير – تعريف متغير نصي – مقارنة متغيرين نصيين – ...) .

✓ **تحديد قيمة النص:**

طبعا القيمة الافتراضية ستحدد في الكونستراكتور (**constructor**) الأول (الذي بدون بـــارميترات (**parameters**))، هذا يمكنك اختيار أي قيمة نصية تعجبك رغم أن القيمة المعتادة (**Welcome to Java**) تعـجبني أكثر من أي قيمة أخرى، أما بالنسبة للكونستراكتور الثاني (الذي يأخذ بارميتر (**parameter**) واحد) فالـــقيمة سيحددها المستخدم.

```
public  SearchProperty ( ) {
        this.text = "Welcome to Java" ;
}
public  SearchProperty ( String ptext ) {
        this.text = ptext ;
}
```

✓ دالة ()search:

الدالة **search()** ستقوم بالبحث عن الكلمة أو العبارة المـــــــطلوبة (**pw**)، إذا وجدتها في النص (**text**) تخبر المستخدم بأن الكلمة المطلوبة (**pw**) مـــــوجودة في النص (**text**)، أما إذا لم تجدها فستعلمه بأن الكلمة المطلوبة (**pw**) لا توجد في النص (**text**)، ومنه، ضروري استعمـــال جملة **if**.

```
public  void  search ( String pw ) {
        if ( this.text.contains ( pw ) ) {
                System.out.println ( "We found " + pw + " in the text." ) ;
        } else {
                System.out.println ( "We do not found " + pw +" in the text." ) ;
        }
}
```

✓ دالة ()contains:

دالة **contains()** من الكـــلاس (**class**) **String** تأخذ بارميترا (**parameter**) نصيـــــا، إذا كانت القيمة المأخوذة موجودة في النص المطبق عليه الدالة فإنها ترجع **true** و العكس صـــحيح.

```
String  text = "Hello World" ;
String  w = "world" ;
if ( text.contains ( w ) ) {
        System.out.println ( "Yes" ) ;
} else {
        System.out.println ( "No" ) ;
}
```

ستذهب كل توقعـــاتك أدراج الرياح ☹، عند تجربة هذا الكود ستظهر كلمة **No**، لأن الكلمة (**w**) موجودة في النص لكن باختلاف الحرف الأول، لأن الدالة حســـــــاسة لطريقة كتابة الحروف هذا

اعتبرها غير موجودة في النص، أظن أن أول ما سيخطر ببالك هو استعمال إحدى الدالـــــــــتين **toLowerCase()** أو **toUpperCase()**.

```
String  text = "Hello World" ;
String  w = "world" ;
if ( text.toLowerCase ( ) .contains ( w.toLowerCase ( ) ) ) {
        System.out.println ( "Yes" ) ;
} else {
        System.out.println ( "No" ) ;
}
```

هنا لن يحدث أي مشكل فقد تم تحويل كل الحروف بشكل واحد ليسهل مقارنتها . ☺

إذن ستتغير الدالة **search()** تغيرا طفيفا:

```
public  void  search ( String pw ) {
        if ( this.text.toLowerCase ( ) .contains ( pw.toLowerCase ( ) ) ) {
                System.out.println ( "We found " + pw + " in the text." ) ;
        } else {
                System.out.println ( "We do not found " + pw +" in the text." ) ;
        }
}
```

✓ تحديد مكان الكلمة:

الآن و بعد أن حددنا وجود الكلمة (**pw**) من عدمها في النص (**text**)، نريد أن نعرف مكانها في هذا النص (**text**)، لدينا طرق عديدة لفعل ذلك لكننا سنحدد مكان الكلمة (**pw**) عبر الأرقام، لذلك سنستعمل الدوال **indexOf()** و **length()**.

✓ تذكير:

في المتغيرات النصية يمثل كل حرف برقم و يبدأ العد من 0، و تعتبر المسافة (**espace**) حرفا أيضا .

✓ دالة ()indexOf:

دالة **()indexOf** من الكلاس (**class**) **String** تأخذ بـــارميترا (**parameter**) نصيا، إذا كان القيمة المأخوذة موجودة في النص المطبق عليه الدالة فإنها ترجع رقم أول حرف منها (القيمة)، و إذا لم تكن موجودة فإنها ترجع (**-1**)، أي في كلتا الحالتين فإنها ترجع عددا صحيحا (**Integer**).

```
String  text = "Welcome to Java" ;
String  w = "come"   ;
System.out.println ( text.indexOf ( w ) ) ;
```

عند تنفيذ هذا الكود سيظهر الرقم (**3**)، لأنه هو رقم أول حرف من الكلمة (**w**) في النـــــص (**text**).

```
String  text = "Welcome to Java" ;
String  w = "cime"   ;
System.out.println ( text.indexOf ( w ) ) ;
```

أما عند تنفيذ هذا الكود سيظهر الرقم (**-1**)، لأن الكلمة (**w**) غير موجودة في النـــــص (**text**).

(أنصحك بتجربة الدالة بنفسك لكي تفهمها جيدا)

✓ تحديد مكان الكلمة بالأرقام:

سيتم تحديد الكلمة بالأرقام على الشكل التالي:

الكلمة المطلوبة توجد بين الحرف كذا و الحرف كذا.

الحرف الأول سنحدده باستعمـــــال دالـــة **()indexOf** أما الحرف الأخير فسنحدده باستعمـــــال الدالـــتين **()indexOf** و **()length**، لأن الحرف الأخير هو رقم الحرف الأول زائد طول الكلمة المطلوبة (**pw**) ناقص واحد. (اقرأ هذه الجملة جيد لتفهمها)

ستتغير الدالة **()search** الآن لتصبح على الشكل التالي:

```
public  void  search ( String pw ) {
        if ( this.text.toLowerCase ( ) .contains ( pw.toLowerCase ( ) ) ) {
```

```
            System.out.println ( "We found " + pw + " in the text." ) ;
            int  fc = this.text.toLowerCase ( ).indexOf ( pw.toLowerCase ( ) ) ;
            int  lc = fc + pw.length ( ) – 1 ;
            System.out.println ( "The word between " + fc + " and " + lc + ".") ;
      } else {
            System.out.println ( "We do not found " + pw +" in the text." ) ;
      }
}
```

✓ مشكل منطقي:

المشكل المطروح الآن عبارة عن مشكلين:

1- المشكل الأول هو أن المستخدم عندما يتعامل مع النص يبدأ العد من 1 و ليس 0، لكن و بما أننا لسنا في برنامج حقيقي فهذا المشكل يمكن تجاوزه إضافة إلى أن حله بسيط.(إضافة العدد واحد للرقمين معا)

2- المشكل الثاني هو الأصعب، عندما يدخل المستخدم كلمة تتكرر عدة مرات في النص، فدالة **indexOf()** ستعتبر الكلمة الأولى فقط.(يمكنك التجربة بنفسك لتتأكد)

✓ مبدأ البحث عن أكثر من كلمة:

الدالة **indexOf()** تعود برقم أول حرف من أول كلمة موجودة في النص، هذا فقط للتوضيح.

كيف يمكننا تغيير الدالة **search()** لتجد جميع الكلمات المكررة في النص ؟؟؟



تمعن في الموضوع جيدا و ستجد الحل بإذن الله...........................

لا تنتقل للصفحة التالية حتى تجد الحل...............................

أو تقترب من إيجاد الحل..

أو تستسلم...

أرى أنك قد وجدت الحل (على ما أظن). ☺

الـــــحل بسيط (صراحة أحب التشويق لهذا جعلتك تتشوق لمعرفة الحـل)، سنستخدم الـ **Loop**، في كل لفة سنبحث في جزء مختلف من النص عن اللفة السابقة و اللفة التي تليها، لهذا سنحتاج الدوال **substring()** و **indexOf()** و **contains()** و **length()**، لفهم المبدأ جيدا سأعطي مثالا بسيطا.

✓ **مثال:**

النص: **Welcome to cool country**.

الكلمة المطلوبة: **co**.

تطبيق مبدأ البحث الجديد:

1- البحث في النص كاملا.

W	e	l	c	o	m	e		t	o		c	o	o	l		c	o	u	n	t	r	y
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

2- البحث في الجزء المتبقي 1.

m	e		t	o		c	o	o	l		c	o	u	n	t	r	y
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

3- البحث في الجزء المتبقي 2.

o	l		c	o	u	n	t	r	y
13	14	15	16	17	18	19	20	21	22

4- البحث في الجزء المتبقي 3.

u	n	t	r	y
18	19	20	21	22

5- ليس هناك أي قيمة مطابقة للكلمة المطلوبة يجب إيقاف البحث.

✓ دالة ()substring:

في الحقيقة هناك دالتي **substring()** و هذا يذكرنا بـ **Overolading**، المهم... هناك دالة تأخذ بارميترا (**parameter**) واحدا و الأخرى تأخذ اثنين و كلاهما تأخذان أعداد صحيحة (**Integer**) و ترجعان نصا (**String**) و تنتميان للكلاس (**class**) **String**، و دور هذه الدالة هو إرجاع مقطع من النص المطبق عليه الدالة، الأولى أي التي تأخذ بارميترا واحدا ترجع من الحرف الذي رقمه يساوي القيمة المأخوذة إلى آخر النص.

```
String  text = "Welcome to Java" ;
System.out.println ( text.substring ( 11 ) ) ;
```

أما الثانية أي التي تأخذ بارميترين ترجع من الحرف الذي رقمه يساوي القيمة الأولى المأخوذة إلى الحرف الذي رقمه يساوي القيمة الثانية المأخوذة ناقص واحد .

```
String  text = "Welcome to Java" ;
System.out.println ( text.substring ( 11 , 15 ) ) ;
```

(أنصحك بتجربة هذه الدالة عدة مرات بنفسك لتفهم طريقة عملها جيدا)

الآن بعد أن فهمت جزءا من كيفية عمل حل لهذا المشكل المطروح، يمكنك كتابة الدالة أو يمكنك الاستعانة بالدالة التالية:

```
public  void  search ( String pw ) {
        if ( this.text.toLowerCase ( ) .contains ( pw.toLowerCase ( ) ) ) {
                System.out.println ( "We found " + pw + " in the text." ) ;
                System.out.println ( "----------------------------------------" ) ;
                int  t = 0 , p = 0 ;
                String  st ;
                for ( int  i = 0 ; i < text.length ( ) ; i++) {
                        st = this.text.toLowerCase ( ).substring ( p ) ;
                        if ( st .contains ( pw.toLowerCase ( ) ) ) {
                                t++ ;
                                p = p + st.indexOf ( pw.toLowerCase ( ) ) + pw.length ( ) ;
                                System.out.println ( "Word n°" + t + " :" ) ;
```

```java
                        System.out.println ( "Between " + ( p - pw.length( ) + 1 ) + " and "
+ ( p - 1 + 1 ) + "." ) ;
                        System.out.println ( "--------------------" ) ;
                  } else {
                        System.out.println ( "Number Of words : " + t ) ;
                        break ;
                  }
            }
      } else {
            System.out.println ( "We do not found " + pw +" in the text." ) ;
      }
}
```

1–تعريف متغيرين:

```java
            int  t = 0 , p = 0 ;
```

(**t**) يمثل عدد مرات تكرار الكلمة في النص.

(**p**) يمثل القيمة التي سنمررها للدالة **substring()**.

2– استعمال الـ **for** و تحديد عدد لفاتها الأقصى:

إن الحد الأقصى لعدد لفات الـ **for** هو طول النص، لاستحالة وجود شرط آخر، تخيل معي نصا طوله 16 حرفا، كلها متشابهة مثلا **a** و مررنا للدالة **search()** هذا الحرف، ستعلمك الدالة أن عدد مرات تكرار هذا الحرف في النص هو 16، مثال آخر، لدينا نص (**hay aye good bay**) طوله 16 حرفا كذلك (يا ها من صدفة هههه) و مررنا للدالة الكلمة **ay**، ستعلمك الدالة بأن عدد مرات تكرار الكلمة هو 3، خلاصة القول أن في جميع الحالات أقصى عدد مرات تكرار قيمة معينة في أي نص هو طول النص نفسه.

3- الـ if و التركيب الطويل العجيب + else:

```
if ( st .contains ( pw.toLowerCase ( ) ) )
```

هذا التـــركيب صحيح بدون شك (عوض قيمة (st) لتفهم الفـكرة) لأن كل الـدوال لها إرجـــــاع، و بالتـــالي سيعوض استدعائهـــــا بقيمـة معينة ستطبق عليها الـــدالة الموالية إلى أن نصـل إلى دالـة **contains()** و التي سترجع إما **true** أو **false** و بالطبع في كلتا الحالتين لن نحتاج لأي معاملات (== أو < أو > أو ...)، لكي تفهم الموضوع جيدا و أتأكد أنك فهمته:

- الدالة **toLowerCase()** سترجع نصا يطابق النص الأصلي (**text**) لكن بحروف صغيرة.
- الدالة **substring()** ستأخذ جزءا محددا من القيمة السابقة (لقد تطرقنا هذه الدالة سابقا).
- ستتأكد الدالة **contains()** من وجود القيمة المررة لها (**pw.toLowerCase()**) في النص الذي أرجعته الدالة **substring()**.
- و كما ذكرت سترجع الدالة **contains() true** أو **false**.
- طبقنا الدالة **toLowerCase()** على القيمة المررة (**pw**) لتوحيد نوعية الحروف.

في اللفة الأولى ستكون قيمة (**p**) هي 0 و بالتالي ستأخذ الدالة **substring()** النص (**text**) كاملا و ستتأكد الدالة **contains()** من وجود الكلمة المطلوبة (**pw**) في النص (**text**)، ثم تزداد قيمة (**p**)، في اللفة الثانية ستأخذ الدالة **substring()** الجزء المتبقي من النص (**text**) و ستتأكد الدالة **contains()** من وجود الكلمة المطلوبة (**pw**) في الجزء المتبقي، إذا كانت الكلمة (**pw**) موجودة في هذا الجزء ستزداد قيمة (**p**) و ستكمل للفة الثالثة أما إذا لم تكن موجودة فتعلم المستخدم بعدد مرات تكرار الكلمة المطلوبة (**pw**) و ستوقف (**else**) الـ **Loop**.

```
else {
        System.out.println ( "Number of words : " + t ) ;
        break ;
}
```

(لكي تفهم هذه الفقرة ارجع للمثال السابق (**Welcome to cool country**))

4- عدد مرات تكرار الكلمة:

```
t++ ;
```

في كل مرة تجد الكلمة المطلوبة ستضيف **1** للمتغير (**t**) الذي يمثل عدد مرات تكرار الكلمة في النص.

5- قيمة (**p**):

```
p = p + st.indexOf ( pw.toLowerCase ( ) ) + pw.length ( ) ;
```

إذا عوضنا (**st**):

```
p = p + this.text.toLowerCase ( ).substring ( p ).indexOf ( pw.toLowerCase ( ) ) + pw.length
( ) ;
```

في اللفة الأولى ستكون قيمة (**p**) هي 0 أي أننا سنتجاهل قيمته، الدالة **indexOf()** ستحدد رقم أول حرف في الكلمة (**pw**) بالنص (**text**) ثم سيضاف إليه طول الكلمة (**pw**)، أي أن قيمة (**p**) ستساوي رقم الحرف الذي بعد آخر حرف في الكلمة (**pw**)، لكي تفهم جيدا سنأخذ المثال السابق (**Welcome to cool country**)، لدينا أول **co** في النص (**Welcome to cool country**)، رقم أول حرف منها **c** هو 3 سيضاف إليه طول الكلمة **co** و سيصبح 5، كي نتجاهل الجزء الذي وجدنا فيه الكلمة **co** و نبحث في جزء جديد بفضل الدالة **substring()** و **contains()**.

نعود الآن إلى الشرح، في اللفة الثانية على مستوى الـ **if** :

```
if ( st .contains ( pw.toLowerCase ( ) ) )
```

إذا عوضنا (**st**):

```
if (this.text.toLowerCase ( ).substring ( p ).contains ( pw.toLowerCase ( ) ) )
```

الدالة **substring()** (ستحذف) الجزء الأول الذي وجدنا فيه أول كلمة (**pw**)، بفضل (**p**)، ثم ستقوم الدالة **contains()** بالبحث عن الكلمة (**pw**) في القيمة التي سترجعها **substring()**.

نعود الآن إلى (**p**):

```
p = p + this.text.toLowerCase ( ).substring ( p ).indexOf ( pw.toLowerCase ( ) )
 + pw.length ( ) ;
```

هنا لن تكون قيمة (**p**) هي 0، بل ستمثل طول النص (المحذوف)، و الباقي يمثل رقم الحرف الذي بعد آخر حرف في الكلمة (**pw**)، لكي تفهم لما أضفنا (**p**)، سنرجع للمثال السابق (**Welcome to cool country**)، لقد وجدنا قبل قليل الرقم 5، الذي سيمرر للدالة **substring()** التي ستعيد القيمة (**me to cool country**)، لاحظ أن الجزء الذي وجدنا فيه الكلمة **co** (حذف)، إذا استعملنا الدالة **indexOf()** هنا، ستعيد الرقم 6، رقم الحرف **c** في الجزء المتبقي، لكننا نريد رقمه في النص كاملا، لهذا سنضيف طول النص (المحذوف)، و سيصبح الناتج 11 و هذا هو رقم الحرف **c** في النص كاملا، لكن لا ننسى إضافة طول الكلمة **co** فيصبح المجموع 13. لهذا السبب كتبنا:

```
System.out.println ( "Between " + ( p - pw.length ( ) + 1 ) + " and " + ( p - 1 + 1 ) + "." ) ;
```

طرحنا طول الكلمة (**pw**) من (**p**) للحصول على رقم أول حرف، و طرحنا 1 من (**p**) للحصول على رقم آخر حرف، لكن لماذا أضفنا 1 لكلا الرقمين ؟ (ارجع للمشكل المنطقي و ستفهم)

✓ ملحوظة:

أشرت فيما سبق عدة مرات لقيمة المتغير (**pw**) بالكلمة (**pw**) أو الكلمة المطلوبة (**pw**)، ربما سيعتبره البعض خطأ، وأنا لا ألومهم، لكني استعملت هذه العبارة لتبسيط الشرح فقط، رغم أنه يبقى معنى منطقيا فالمتغير (**pw**) نوعه نص (**String**) و النص يتكون من كلمات، على كل الأحوال تبقى هذه الملحوظة موجهة لمن قرأ حل الإشكالية (الكتاب كاملا). ☺

نبذة عن الكاتب

إسماعيل غلال، من مواليد 1998 بالمغرب، تلميذ بالسلك الثانوي التأهيلي، بالجذع المشترك العلمي.

بقلم الكاتب:

بدأت البرمجة في صيف سنة 2013، و أعجبت بها كثيرا، لدرجة أنه أردت تعلم جميع اللغات دفعة واحدة، و قد سبب لي هذا مشكلا، دخلت في HTML و CSS و PHP و JAVA و JAVASCRIPT و SQL، و لم أتعلمها كلها حتى الآن، لذلك لكل من يريد تعلم لغات البرمجة، فليبتعد عما فعلته أنا، و يتعلم بالتدرج، كما قال الأستاذ عبد الله عيد، التدرج سنة كونية، تعلم لغة بلغة، و لا تخلط بينها أثناء تعلمها .

خاتمة

في الختام، أود أن أهنئ نفسي، لأني أنجزت أول عمل أدبي برمجي لي، و ربما الأخير. أفكر في أن أكمل كتابة أمثلة أخرى عن النصوص و الأعداد و الـ **boolean** و ... لكنني سأعتمد على تقييمك لهذا الكتاب كشرط في فعل هذا، بفضلك ستستمر سلسلة الأمثلة التطبيقية و بفضلك ستتوقف، هناك مسألة اتخاذ القرار شخصيا لكني لا أكتب لأفيد نفسي، بل لأفيد الآخرين و يفيدوني، أحاول أن أكون منتجا و لو بأي شكل كان، يعني صراحة لا أود التطرق لموضوع الاحتكار الآن، و لا التعصب الفكري كذلك، المهم، خلاصة الكلام، لكل من قرأ هذا الكتاب، و لديه أي ملاحظات عنه (سيئاته – حسناته – فكرة لتحسينه - ...)، بريدي الإلكتروني تحت تصرفكم:

ismailghilal@gmail.com

و أشكر مقدما كل من قدم أي ملاحظة، و أشكر أيضا كل من قرأ هذا الكتاب، و أشكر جدا الأستاذ عبد الله عيد، و في الأخير، أود أن أتأسف عن أي خطأ أو نسيان، فإن سهوت فمني و من الشيطان و إن أفلحت فمن الله عز و جل.

و السلام عليكم و رحمة الله تعالى